SAND REPORT

SAND2001-3769
Unlimited Release
Printed July 2002

Kernel Near Principal Component Analysis

Shawn Martin

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

2001-3769
Unlimited Release
Printed July 2002

Kernel Near Principal Component Analysis

Shawn Martin
Computational Biology
Sandia National Laboratories PO Box 5800
Albuquerque, NM 87185-0318
smartin@sandia.gov

July 17, 2002

Abstract

We propose a novel algorithm based on Principal Component Analysis (PCA). First, we present an interesting approximation of PCA using Gram-Schmidt orthonormalization. Next, we combine our approximation with the kernel functions from Support Vector Machines (SVMs) to provide a nonlinear generalization of PCA. After benchmarking our algorithm in the linear case, we explore its use in both the linear and nonlinear cases. We include applications to face data analysis, handwritten digit recognition, and fluid flow.

Keywords: nonlinear Principal Component Analysis, Support Vector Machine kernels, Gram-Schmidt

Acknowledgement

This work was supported by MICS project 547 task 01.11.

Contents

Figures

Kernel Near Principal Component Analysis

1 Introduction

The goal of Principal Component Analysis (PCA) is to find a coordinate representation for a given data set such that the most variance in the data is captured in the least number of coordinates. This representation is typically found by performing a linear transformation of the original data via the Singular Value Decomposition (SVD). The resulting singular vectors provide an orthonormal basis for the data while the singular values provide information on the importance of each basis vector. A review of PCA, its history, examples of applications, and information about the SVD can be found in [13], [15], [7], and [26]. Some of the original papers on PCA include [19] and [10].

In addition to the standard linear version of PCA, some nonlinear variants have been proposed. These methods include Hebbian networks, multi-layer perceptrons, Principal Curves, and kernel PCA. The linear PCA neuron [18], [7] in a Hebbian network can be replaced by a nonlinear neuron to perform nonlinear PCA; a multi-layer perceptron having enough layers and using nonliner activation functions can be considered a type of nonlinear PCA [16], [7]; Principal Curves [9] pass through the "middle" of a given data set and can be shown to be principal components when constrained to be linear; and kernel PCA [24] performs linear PCA after implicitly remapping the original data by a nonlinear function. In addition, there are other methods in data analysis similar in spirit to PCA. These include Projection Pursuit [8] and Independent Component Analysis [14], [11].

In this paper we propose another version of PCA. Our version includes a modification of linear PCA coupled with a nonlinear extension. In our modification of linear PCA we locate a basis using the same criterion employed by PCA but subject to an additional constraint. Our basis is required to correspond directly to a linearly independent subset of our original data. Essentially, we find an ordered linearly independent subset of our data which best approximates the PCA expansion when used with Gram-Schmidt orthonormalization. We call this approximation *near PCA* (nPCA).

Our nonlinear extension of nPCA is based on the use of Support Vector Machine (SVM) type kernels. SVM kernels allow us to remap our data implicitly using nonlinear functions. Following such remappings we apply nPCA. Hence our nonlinear version of PCA is called *kernel near PCA* (knPCA).

Our paper is organized as follows: in Section 2 we provide the background necessary to describe knPCA; in Section 3 we give the actual description of knPCA; in Section 4 we benchmark knPCA using the linear kernel; in Section 5 we provide some applications and examples of knPCA; in Section 6 we discuss the properties of knPCA and compare it with other algorithms for nonlinear PCA; in Section 7 we describe how knPCA could be modified/extended for other uses; and in Section 8 we conclude the paper.

2 Background

KnPCA is a combination of Gram-Schmidt orthonormalization and PCA all rewritten in terms of inner products so that SVM type kernel functions can be used. We therefore provide some background on Gram-Schmidt, PCA, and SVM kernel functions.

Gram-Schmidt

Gram-Schmidt orthonormalization [26] is a procedure for transforming a set of linearly independent vectors $\{\mathbf{x}_1, \ldots \mathbf{x}_m\}$ into an orthonormal basis $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$. This basis is constructed iteratively via projetions. Specifically,

$$\begin{gathered}
\mathbf{u}_1 = \tfrac{\mathbf{x}_1}{\|\mathbf{x}_1\|},\ \mathbf{p}_1 = (\mathbf{x}_2, \mathbf{u}_1)\mathbf{u}_1 \\
\mathbf{u}_2 = \tfrac{\mathbf{x}_2 - \mathbf{p}_1}{\|\mathbf{x}_2 - \mathbf{p}_1\|},\ \mathbf{p}_2 = (\mathbf{x}_3, \mathbf{u}_2)\mathbf{u}_2 + (\mathbf{x}_3, \mathbf{u}_1)\mathbf{u}_1 \\
\vdots \\
\mathbf{u}_m = \tfrac{\mathbf{x}_m - \mathbf{p}_{m-1}}{\|\mathbf{x}_m - \mathbf{p}_{m-1}\|},
\end{gathered}$$

where we use $(\mathbf{x}, \mathbf{y})$ to denote the inner product (dot product) of $\mathbf{x}$ with $\mathbf{y}$. Now we can represent our original data $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$ in the new basis as the upper triangular matrix

$$\begin{pmatrix} (\mathbf{x}_1, \mathbf{u}_1) & \cdots & (\mathbf{x}_m, \mathbf{u}_1) \\ \vdots & \ddots & \vdots \\ (\mathbf{x}_1, \mathbf{u}_m) & \cdots & (\mathbf{x}_m, \mathbf{u}_m) \end{pmatrix}.$$

We can also include linearly dependent data $\{\mathbf{x}_{m+1}, \ldots, \mathbf{x}_n\}$ as additional columns in the matrix

$$U = \begin{pmatrix} (\mathbf{x}_1, \mathbf{u}_1) & \cdots & (\mathbf{x}_m, \mathbf{u}_1) & (\mathbf{x}_{m+1}, \mathbf{u}_1) & \cdots & (\mathbf{x}_n, \mathbf{u}_1) \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (\mathbf{x}_1, \mathbf{u}_m) & \cdots & (\mathbf{x}_m, \mathbf{u}_m) & (\mathbf{x}_{m+1}, \mathbf{u}_m) & \cdots & (\mathbf{x}_n, \mathbf{u}_m) \end{pmatrix}.$$

PCA

PCA is typically formulated as an eigenvalue problem which is closely related to the SVD [15], [7]. It also typically described as a procedure for successively capturing the maximal variance in the data. The PCA eigenvectors (singular vectors) satisfy [15]

$$\begin{array}{lll} \mathbf{u}_1 & \text{maximizes} & \sum_{i=1}^n (\mathbf{x}_i, \mathbf{u}_1)^2 \\ \mathbf{u}_2 & \text{maximizes} & \sum_{i=1}^n (\mathbf{u}_2, \mathbf{x}_i - (\mathbf{x}_i, \mathbf{u}_1)\mathbf{u}_1)^2 \\ & \vdots & \\ \mathbf{u}_m & \text{maximizes} & \sum_{i=1}^n (\mathbf{u}_m, \mathbf{x}_i - \sum_{j=1}^{m-1} (\mathbf{x}_i, \mathbf{u}_j)\mathbf{u}_j)^2 \end{array}$$

where $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ are also required to be orthonormal. (When we write "$\mathbf{u}_1$ maximizes $\sum_{i=1}^n (\mathbf{x}_i, \mathbf{u}_1)^2$," we mean $\mathbf{u}_1 = \mathbf{u}$ such that $\mathbf{u}$ maximizes $\sum_{i=1}^n (\mathbf{x}_i, \mathbf{u})^2$, with similar meanings for $\mathbf{u}_2, \ldots, \mathbf{u}_m$.)

PCA is often illustrated by finding the major and minor axes in a cloud of data filling an ellipse. The first eigenvector corresponds to the major axis of the ellipse while the second eigenvector corresponds to the minor axis. This example is shown later in Figure 1.

SVM Kernels

SVMs (Support Vector Machines) [6], [4], [28] are classifiers which use kernels from integral operators to remap data implicitly via nonlinear functions [3], [4]. A function $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a kernel for a nonlinear map $\Phi : \mathbb{R}^n \to F$ if

$$k(\mathbf{x}, \mathbf{y}) = (\Phi(\mathbf{x}), \Phi(\mathbf{y})),$$

where F is a Hilbert space.

Such nonlinear maps can be useful in classification by transforming data which is not linearly separable in the original space into linearly separable data in some higher dimensional space. This technique is well illustrated when applying the map $\Phi(x, y) = (x^2, \sqrt{2}xy, y^2)$ to the planar two class exclusive-or problem. In this problem the first class consists of two clusters, one centered about the point $(1, 1)$ and the other centered about $(-1, -1)$. The second class consists of two additional clusters, one centered about $(1, -1)$ and another centered about $(-1, 1)$.

The kernels corresponding to the nonlinear maps are computational devices. They allow the use of the nonlinear maps while avoiding calculations in their high (possibly infinite) dimensional ranges. Any algorithm which can be written in terms of inner products can use the kernel functions. When the inner products in the algorithm are replaced by kernels, the algorithm is effectivley transported into the range of some nonlinear map.

Some examples of kernels are

- the polynomial kernel $k(\mathbf{x}, \mathbf{y}) = ((\mathbf{x}, \mathbf{y}) + c)^d,\ c \geq 0,\ d \in \mathbb{Z}_{>0}$,
- the radial basis function (RBF) kernel $k(\mathbf{x}, \mathbf{y}) = e^{\frac{-\|\mathbf{x}-\mathbf{y}\|^2}{2\sigma^2}},\ \sigma \neq 0$,
- the neural network kernel $k(\mathbf{x}, \mathbf{y}) = \tanh(a(\mathbf{x}, \mathbf{y}) + b),\ a, b \geq 0$.

For more on kernel functions see [4], [5].

3 Algorithm

Having provided the background on Gram-Schmidt, PCA, and SVM kernels, we can describe the basic strategy of knPCA. First, a linearly independent subset of our data is chosen using the PCA criterion. Next, a kernel version of Gram-Schmidt is performed using that subset. The implementation of this strategy involves rewriting Gram-Schmidt in terms of inner products and constraining the PCA criterion so that each $\mathbf{u}_i$ corresponds directly to an actual data point. This is nPCA. By replacing inner products with kernels we get knPCA.

KnPCA is based on a reformulation of the Gram-Schmidt procedure. To describe this reformulation we observe that Gram-Schmidt is recursive and that we can rewrite it as follows [17]

$$\begin{aligned}
(\mathbf{x}_1, \mathbf{u}_1) &= \|\mathbf{x}_1\| = \tfrac{(\mathbf{x}_1,\mathbf{x}_1)}{(\mathbf{x}_1,\mathbf{u}_1)} \\
(\mathbf{x}_2, \mathbf{u}_1) &= \tfrac{(\mathbf{x}_2,\mathbf{x}_1)}{\|\mathbf{x}_1\|} = \tfrac{(\mathbf{x}_2,\mathbf{x}_1)}{(\mathbf{x}_1,\mathbf{u}_1)} \\
(\mathbf{x}_2, \mathbf{u}_2)^2 &= \|\mathbf{x}_2 - \mathbf{p}_1\|^2 = \|\mathbf{x}_2\|^2 - \|\mathbf{p}_1\|^2 \\
(\mathbf{x}_2, \mathbf{u}_2) &= \tfrac{1}{(\mathbf{x}_2,\mathbf{u}_2)} \left[(\mathbf{x}_2, \mathbf{x}_2) - (\mathbf{x}_2, \mathbf{u}_1)^2\right] \\
(\mathbf{x}_3, \mathbf{u}_1) &= \tfrac{(\mathbf{x}_3,\mathbf{x}_1)}{(\mathbf{x}_1,\mathbf{u}_1)} \\
(\mathbf{x}_3, \mathbf{u}_2) &= \tfrac{1}{(\mathbf{x}_2,\mathbf{u}_2)} \left[(\mathbf{x}_3, \mathbf{x}_2) - (\mathbf{x}_2, \mathbf{u}_1)(\mathbf{x}_3, \mathbf{u}_1)\right] \\
(\mathbf{x}_3, \mathbf{u}_3) &= \tfrac{1}{(\mathbf{x}_3,\mathbf{u}_3)} \left[(\mathbf{x}_3, \mathbf{x}_3) - ((\mathbf{x}_3, \mathbf{u}_2)^2 + (\mathbf{x}_3, \mathbf{u}_1)^2)\right],
\end{aligned}$$

and in general

$$(\mathbf{x}_i, \mathbf{u}_j) = \frac{1}{(\mathbf{x}_j, \mathbf{u}_j)} \left[(\mathbf{x}_i, \mathbf{x}_j) - \sum_{k=1}^{j-1} (\mathbf{x}_j, \mathbf{u}_k)(\mathbf{x}_i, \mathbf{u}_k)\right].$$

This formula can also be used for the remainder of our (linearly dependent) data $\{\mathbf{x}_{m+1}, \ldots, \mathbf{x}_n\}$ to arrive at the matrix U in the Section 2.

Another useful addition to Gram-Schmidt is a change of basis matrix for switching from $\{\mathbf{u}_1, \ldots, \mathbf{u}_m\}$ to $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$. This matrix, $T = (\mathbf{t}_1, \ldots, \mathbf{t}_m)$, can be computed columnwise by

$$\begin{aligned}
\mathbf{t}_1 &= \tfrac{1}{(\mathbf{x}_1,\mathbf{u}_1)} \mathbf{e}_1 \\
\mathbf{t}_2 &= \tfrac{1}{(\mathbf{x}_2,\mathbf{u}_2)} \left[\mathbf{e}_2 - (\mathbf{x}_2, \mathbf{u}_1)\mathbf{t}_1\right] \\
&\vdots \\
\mathbf{t}_m &= \tfrac{1}{(\mathbf{x}_m,\mathbf{u}_m)} \left[\mathbf{e}_m - \textstyle\sum_{i=1}^{m-1} (\mathbf{x}_m, \mathbf{u}_i)\mathbf{t}_i\right],
\end{aligned}$$

where $\{\mathbf{e}_1, \ldots, \mathbf{e}_m\}$ are the standard basis vectors. Now TU expresses our data in terms of the basis $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$.

This formulation of Gram-Schmidt has two principal advantages over the standard formulation for use with knPCA. First, it is expressed in terms of inner products to allow the use of SVM kernels. Second, the change of basis matrix T allows us to express our data in terms of actual examples in our data set. This will allow us to interpret the results of any nonlinear remapping of our data.

We next modify our reformulation of Gram-Schmidt by a constrained version of the PCA criterion. Specifically, we use the PCA criterion to select the linearly independent subset used in Gram-Schmidt. Abandoning our previous labeling $\{\mathbf{x}_1, \ldots, \mathbf{x}_m\}$

in favor of the more accurate labeling $\{\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_m}\}$, this subset is selected by

$$\begin{array}{lll} \mathbf{x}_{i_1} & \text{maximizes} & \left\{\frac{1}{\|\mathbf{x}_i\|^2}\sum_{j=1}^n (\mathbf{x}_i, \mathbf{x}_j)^2\right\}_{i=1}^n \\ \mathbf{x}_{i_2} & \text{maximizes} & \left\{\frac{1}{\|\mathbf{x}_i\|^2}\sum_{j=1}^n \left[(\mathbf{x}_i, \mathbf{x}_j) - (\mathbf{x}_i, \mathbf{u}_1)(\mathbf{x}_j, \mathbf{u}_1)\right]^2\right\}_{i=1}^n \\ & \vdots & \\ \mathbf{x}_{i_m} & \text{maximizes} & \left\{\frac{1}{\|\mathbf{x}_i\|^2}\sum_{j=1}^n \left[(\mathbf{x}_i, \mathbf{x}_j) - \sum_{l=1}^{m-1}(\mathbf{x}_i, \mathbf{u}_l)(\mathbf{x}_j, \mathbf{u}_l)\right]^2\right\}_{i=1}^n, \end{array}$$

where the inner products $(\mathbf{x}_\bullet, \mathbf{u}_l)$ are computed using the reformulation of the Gram-Schmidt procedure described above. (For an explanation of the "maximizes" notation see Section 2.)

This combination of Gram-Schmidt and PCA is nPCA (near PCA). NPCA provides an interesting approximation of PCA because it selects points in the actual data set with properties similar to those of PCA. In addition, nPCA is readily generalized to knPCA (kernel near PCA) by replacing inner products with kernels. KnPCA then combines the advantages of our reformulation of Gram-Schmidt (kernel use and interpretation after remapping) with the approximate statistical properties of nPCA.

4 Benchmarks

We first benchmark knPCA using the linear kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{y})$. In this case knPCA reduces to nPCA. For benchmarks, we use toy examples, an example to test the numerical stability of nPCA, and an application to face data analysis.

Our first test uses two toy examples in the plane. We compare the singular vectors found by PCA with those found by nPCA on the ellipse filling data cloud mentioned in Section 2, and on a parabola with gaussian noise. These examples are shown in Figure 1.

We next test nPCA for numerical stability on an example found in [26]. In this example we construct an 80×80 matrix $A = U\Sigma V^T$, where U and V are random orthogonal matrices, and Σ is a diagonal matrix with entries $2^{-1}, 2^{-2}, \ldots, 2^{-80}$. We use Gram-Schmidt, Modified Gram-Schmidt, the SVD, and nPCA to find the singular values of A (for Gram-Schmidt and Modified Gram-Schmidt, the diagonal entries in R of the QR decomposition can be used to approximate the singular values [26]). The results, shown in Figure 2, show that nPCA compares favorably to SVD and Modified

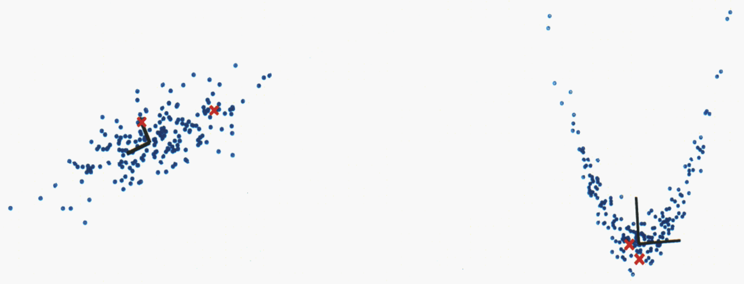

Figure 1. Ellipse and Parabola. In this illustration we compare the singular vectors found by PCA with those found by nPCA for an ellipse (left) and a parabola (right). In both examples, a PCA singular vector is shown as a solid line, while an nPCA singular vector (an actual point in the data set) is marked with an X.

Gram-Schmidt, although it stops before finding all 80 dimensions. (NPCA stops early because the singular values found by nPCA must be obtained by projecting onto actual data points. These values will be less than the true singular values, so nPCA reaches machine precision before either SVD or Modified Gram-Schmidt. Finally, since nPCA stops when it has captured the data to machine precision, it stops before finding all 80 dimensions.)

In [26], this example is used to demonstrate that Modified Gram-Schmidt is stable, while the classical version is not. We use this example to provide evidence (but not proof) than nPCA is stable, despite the fact that it is based on classical Gram-Schmidt. Apparently, our modifications of Gram-Schmidt were sufficient to provide additional stability, thouh we do not investigate this claim further.

In our final example, we compare nPCA to PCA using a database of faces (formerly the ORL database of faces) from AT&T Laboratories in Cambridge, UK [2], [20]. This database consists of four hundred 112×92 gray-scale images of forty people in various poses. We performed both PCA and nPCA on the images to obtain the eigenfaces and near eigenfaces in Figure 3. Although difficult to interpret, it is worth noting that the faces are comparable. In the first PCA eigenface, the actual face is light gray with the hint of a beard while the background progresses from black at the bottom to dark gray at the top. In comparison, the first nPCA eigenface is light, the second has a dark background and beard, and the third nPCA eigenface has a background with a light upper half and a dark lower half. In the second PCA eigenface, glasses can be

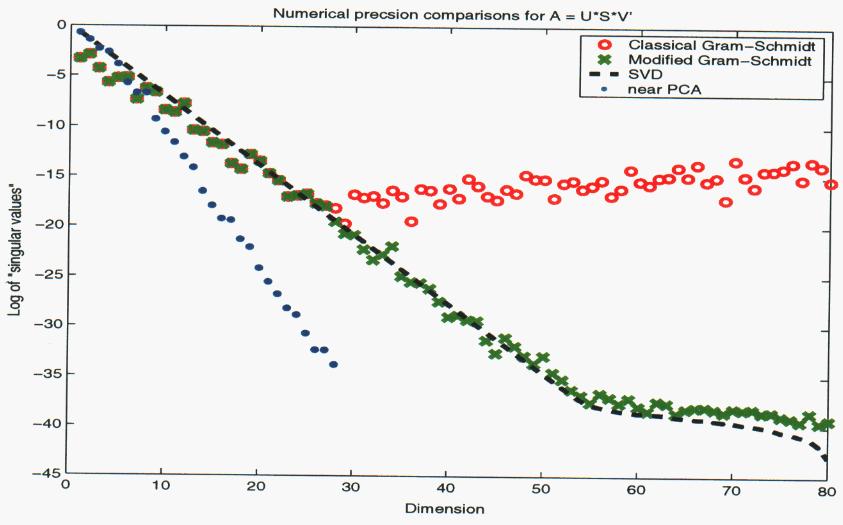

Figure 2. Numerical Stability of nPCA. Here we compare the stability of Gram-Schmidt (GS), Modified Gram-Schmidt (Mod. GS), SVD, and nPCA when finding the singular values of a matrix A, where $A = U\Sigma V^T$, with U and V random orthognal matrices, and Σ a diagonal matrix having entries from 2^{-1} to 2^{-80}.

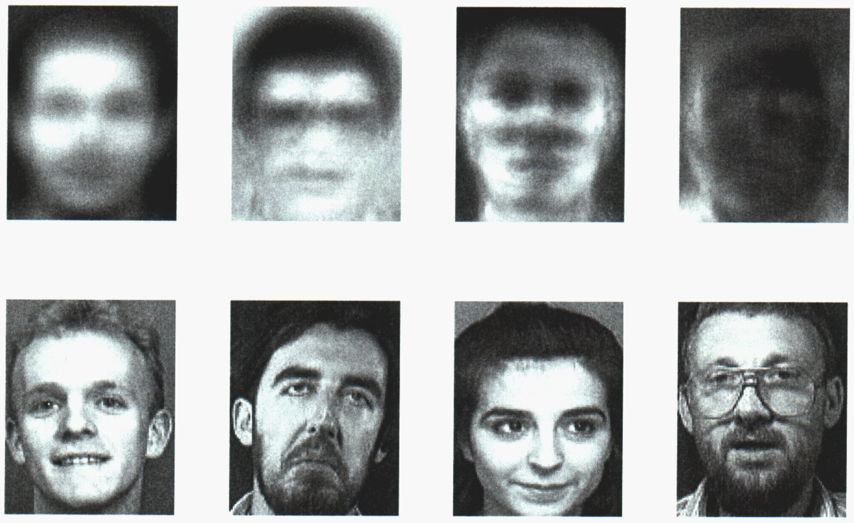

Figure 3. Face Data Analysis. In this figure we compare PCA to nPCA using the AT&T database of faces. On the top row are the first four eigenfaces from standard PCA. On the bottom row are the first four near eigenfaces from nPCA.

seen. In comparison, the fourth nPCA eigenface has glasses. Finally, the position of the eyes can be seen to play a role in both the PCA and nPCA eigenfaces.

As a final test, we added the top three PCA eigenfaces back into the data set and re-ran nPCA on this augmented data set. NPCA returned, as expected, the PCA eigenfaces as its top choices.

5 Applications

In this section we investigate the nonlinear aspects of knPCA and include some applications to larger data sets. To provide some insight into knPCA, we first analyze two more toy problems, this time using polynomial and RBF kernels. Following these examples, we give applications to problems in handwritten digit recognition and fluid flow analysis.

Our first nonlinear toy example is another parabola with Gaussian noise. In this parabola, x-values are drawn from a uniform distribution on $[-1, 1]$ and y-values are the squares of the x-values plus Gaussian noise with a standard deviation of .2. This example was also used in [24] so provides a comparison with that work. Following [24] we use polynomial kernels with $c = 1$ and $d = 2$, 3, and 4. Our results are shown in Figure 4. In the cases $d = 2$ and $d = 3$ the first two singular vectors point in the directions of the arms of the parabola and the third singular vector points in the direction of the noise. When $d = 4$, the first two singular vectors point in the directions of the parabola arms, the second two point in the directions of the noise in the arms, and the fifth points in the direction of overall noise.

Another toy example [22], found in an expanded version [21] of [24], consists of three Gaussian clusters in the region $[-1, 1] \times [-.5, 1]$. Each cluster has a standard deviation of .1. In this example we use knPCA with an RBF kernel (following [21]) of width $\sigma = .22$. Our results are shown in Figure 5. Since we are using a Gaussian kernel which matches our Gaussian clusters, knPCA performs center selection. This occurs because the three centers given by the first three singular vectors are most representative of the data when viewed through the eyes of our Gaussian kernel. This is illustrated by both the singular vectors, which are seen to be centers, and the singular values, which are dominated by the first three nearly equal values.

Our first application is to handwritten digit recognition. We use a United States Postal Service (USPS) database containing 16×16 gray-scale images of handwritten digits from 0 to 9. Of the 9298 images in the database, 7291 are set aside for training

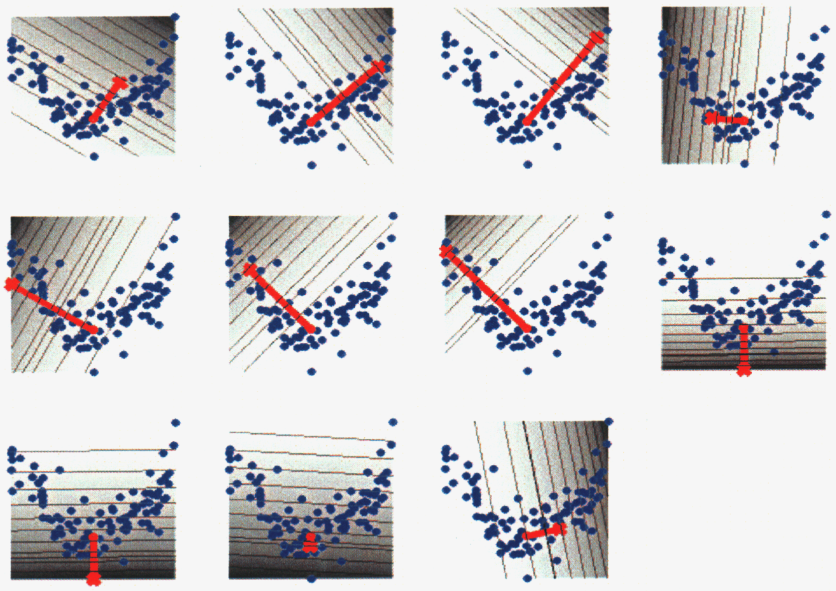

Figure 4. Parabola Revisited. Illustrated here are the singular vectors found by knPCA using the polynomial kernel with $c = 1$ and $d = 2$,, 3 and 4 for a parabola. The first column (from top to bottom) shows the first three singular vectors when $d = 2$, the second column shows the first three singular vectors when $d = 3$, and the third and fourth columns show the first five singular vectors when $d = 4$. In each plot, the origin is marked with a circle, the singular vector is marked with an X, and there is a line connecting the two. In addition, contours are shown that correspond to evenly spaced hyperplanes in the remapped data space, where the hyperplanes are perpendicular to the (remapped) singular vector. These contours are scaled according to the singular value energy (percentage of all singular values), with the actual singular value energy for the given singular vector labeled and shown as a thicker contour.

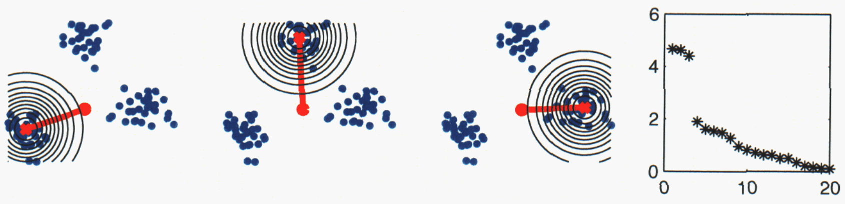

Figure 5. Clusters. Here we show singular vectors and values for three Gaussian clusters found using knPCA with an RBF kernel. The first three plots show the singular vectors and the last plot shows the singular values. The singular vectors are displayed as in Figure 4, with the origin marked by a circle, each singular vector marked with an X, and contours shown corresponding to hyperplanes in the remapped data space.

and 2007 are set aside for testing. This database was first used with SVMs in [23] and later in [24]. It can be downloaded from [27].

Our main interest in this database is the use of knPCA to visualize how the kernels in SVMs yield good classifiers. With this in mind, we used the kernels and paramters found in [23] with knPCA. Specifically, we used nPCA, knPCA with an RBF kernel of width $\sigma = 10$, and knPCA with the kernel $k(\mathbf{x}, \mathbf{y}) = (\frac{1}{256}(\mathbf{x}, \mathbf{y}))^3$. Our results are shown in Figure 6. The plots in Figure 6 suggest that the nonlinear kernels project different digits onto different axes during remapping. This is a desirable situation for a classifier and might explain why nonlinear SVMs work better than linear SVMs on the USPS database. In particular, the polynomial kernel seems to provide the best such projections, corresponding to the fact that the polynomial SVM best classifies the handwritten digits. Using SVM^{light} [12] to train SVMs we obtained a 4.5% error rate using the polynomial classifier, a 5.1% error rate using the RBF classifier, and a 10% error rate using the linear SVM. (We had to divide data by 17 before training the linear SVM.) We also trained linear SVMs on the polynomial and RBF knPCA representations. Using the first 256 coordinates (the first 256 rows of the matrix U), we obtained a 6.1% error rate with the polynomial representation and a 6.6% error rate with the RBF representation. Since knPCA is in no way optimized for classification, it is not surprising that our error rates were higher. It is worth noting, however, that our error rates were similar to the corresponding nonlinear SVMs, and of course lower than the linear SVMs.

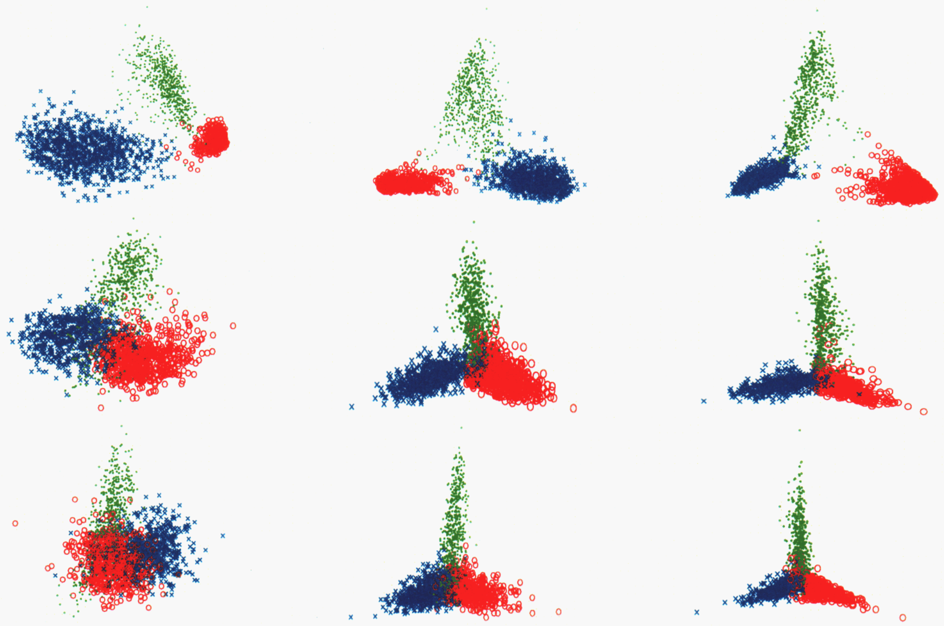

Figure 6. USPS Handwritten Digits. In this figure we compare representations of handwritten digits using nPCA (left column), knPCA with an RBF kernel (center column), and knPCA with a polynomial kernel (right column). In each plot we display three different digits using representations corresponding to these digits. In the upper left plot, for example, we display the digits 0, 1, and 7 using the 1st, 2nd, and 5th coordinates (1st, 2nd, and 5th rows of the matrix U) of the nPCA expansion. We use these coordinates because the corresponding nPCA eigendigits are 1, 0, and 7. In the top row, points marked with an X correspond to the digit zero, points marked with an 0 correspond to the digit one, and points marked with a ● correspond to the digit seven. The second row uses the same scheme with the digits 3, 4, and 6, and the last row uses the digits 2, 5, and 9.

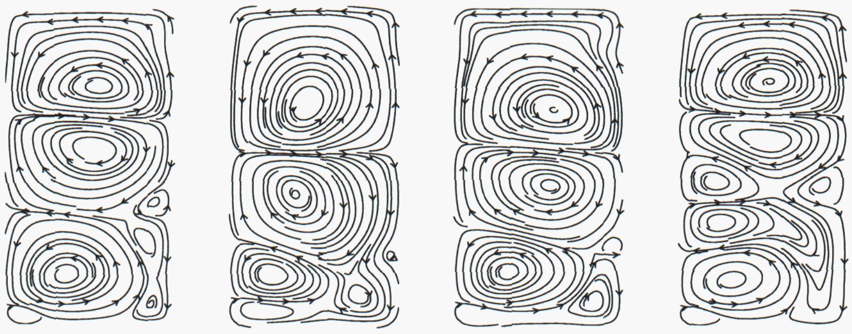

Figure 7. Taylor-Couette Flow. Shown here are the first four nPCA eigenflows from left to right. Each plot shows streamlines of the flow based on the radial and axial velocities.

Our second application is to Taylor-Couette fluid flow analysis. Taylor-Couette flow occurs in fluid trapped between concentric cylinders. When the cylinders are rotated independently, the resulting flow often contains toroidal vortices known as Taylor-Couette cells. Our Taylor-Couette data was generated by numerical simulation [1], [25] and consists of 799 cross-sectional snapshots of the flow as time progresses. Each snapshot contains three velocities and one pressure at points on a 49×21 dimensional grid. We reshaped each snapshot into a 4116 dimensional vector and put the data into a 4116×799 matrix. We performed nPCA on this matrix to obtain the first four eigenflows shown in Figure 7.

Our first remark concerning these eigenflows is that the corresponding singular values account for 96% of the energy of the flow. In other words, the first four singular values added together make up 96% of the sum of all the singular values. This means that the eigenflows under consideration characterize the flow to a high degree of accuracy.

Next we remark that when we project the entire flow onto these eigenflows, we get periodic graphs, as shown in Figure 8. Based on our previous remark, we may conclude with high confidence that the flow under consideration is periodic.

Finally, we remark on the eigenflows themselves. These particular snapshots of the flow represent what the flow is doing most of the time. In this case, the flow consists of three Taylor-Couette cells. The upper cell is stable while the lower cells periodically grow and shrink, at some point even spawning and re-absorbing smaller

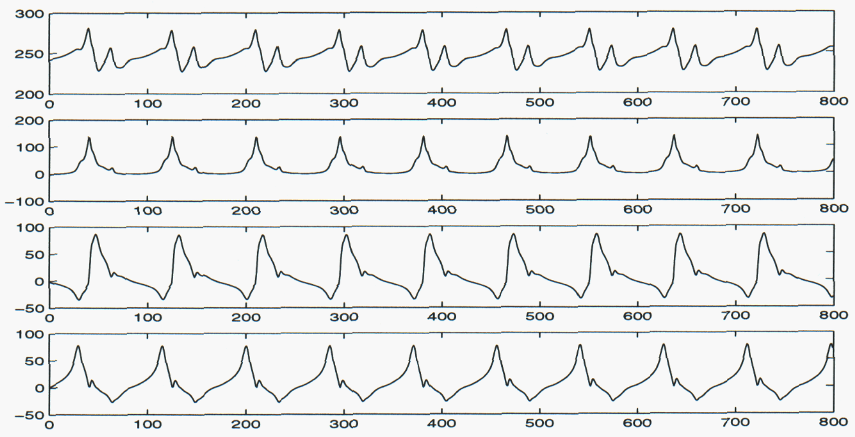

Figure 8. Taylor-Couette Projections. These plots show the projections of the entire Taylor-Couette flow onto the first four eigenflows. The top plot is for the projection onto the first eigenflow, the next plot is for the projection onto the second eigenflow, et cetera. In each plot the x-axis represents time and the y-axis the value of the projection.

sub-cells. All these events are captured by the eigenflows in Figure 7. In fact, by comparing Figures 7 and 8, it is almost possible to visualize the entire flow, including the appearance and disappearance of the sub-cells in the fourth eigenflow.

6 Discussion

Having made a case for the utility of knPCA we now disucss its relation to PCA, its computational complexity, and how it compares with other methods of nonlinear PCA.

Relation to PCA

KnPCA is based on an approximation of PCA. Consequently, knPCA has approximately the same properties as PCA. Some of these properties are [15]:

- maximization of the statistical variance,
- minimization of the mean square truncation error,
- maximization of the mean squared projection,
- minimization of entropy.

In the case of nPCA these properties are directly approximated. In the case of knPCA these properties apply (approximately) after the nonlinear remapping.

In practice, PCA is often used for low dimensional representation and for visualization of data. KnPCA inherits these attributes with the additional advantage of the interpretability of the knPCA singular vectors. This advantage was illustrated in Section 5 with the applications of knPCA to the USPS handwritten digit data and to the Taylor-Couette simulation data. In the case of the USPS database, we were able to produce insightful plots because we knew both which coordinates were most important *and* which digits corresponded to these coordinates. With standard PCA the latter information would be missing. In the case of the Taylor-Couette data we were able to produce actual instances in the flow which best represented the entire flow. This provided a very useful visualization of the flow, especially in the case of the sub-cells in the fourth nPCA eigenflow.

Computational Complexity

KnPCA is straightforward to implement, requiring only a page of `MATLAB` code, and fast enough to use on reasonably large data sets. We used a `MATLAB` version of knPCA for both the AT&T database of faces and the Taylor-Couette flow. Both computations took an hour on a SUN UltraSparc II computer. For the USPS database of handwritten digits we used a `C++` code running on a Pentium III computer. The USPS runs took 2 days each (computations were stopped after 256 eigendigits were found). The disparity in these calculation times is not due to the computer speeds, but rather to the number of data points in the various applications. The AT&T database contained 400 points, the Taylor-Couette flow 799 points, and the USPS database 7291 points.

To explain these disparities further, we point to the fact that knPCA spends most of its time computing variances. This is an $O(qmn^2 + m^2n^2)$ computation, where q is the ambient dimension of the problem (each data point is in $\mathbb{R}^q$), m is the number of singular vectors computed, and n is the number of data points. In our applications, we reduced this computation to $O(m^2n^2)$ by using a kernel cache, but the n^2 dependence still caused disporportionately slower times when using the much larger USPS data set.

It is also worth noting that the knPCA algorithm has room for improvement. First of all, the algorithm contains many redunandant calculations. If even the most obvious of these redundancies (recomputing the projections onto the known singular vectors during each iteration) were eliminated, the algorithm would be $O(mn^2)$. Second, the algorithm is parallelizable. Third, the kernel cache mentioned previously is unnecessary. This may or may not be an improvement depending on the computer being used, but allows a trade off between processor speed and memory use.

Comparisons

Hebbian network algorithms can be used for computing principal components [18], [7]. These algorithms can then be modified using nonlinear activaction functions to compute nonlinear principal components. These components, however, are difficult to interpret both in theory and in practice. KnPCA, on the other hand, produces nonlinear principal components which have a definite geometric explanation and are also interpretable in practice.

Multi-Layer Perceptrons can be used to extract nonlinear principal components by training the identity function on the data using a five layer bottleneck architecture with nonlinear activation functions [16], [7]. The bottleneck layer yields the nonlinear principal components. These components are also difficult to interpret. In addition, the number of desired components must be decided upon before using the algorithm, and the algorithm itself requires nonlinear optimization. (With nonlinear optimization there is the possibility of becoming trapped in a local minima.) In comparison, knPCA can find any number of nonlinear principal components and uses a deterministic, essentially linear (after nonlinear remapping) algorithm.

Principal Curves are smooth curves which pass through the "middle" of a data set [9]. They are computed iteratively (requiring nonlinear optimization) by averaging nearby points in the data set. Principal Curves correspond to principal components when constrained to be linear so that they are a generalization of PCA with a direct geometric interpretation. Principal Curves have been generalized to surfaces but there exist no higher dimensional analogues. In comparison, knPCA is a more direct generalization of PCA (requiring only linear optimization) and can be used to produce higher dimensional representations.

Kernel PCA is very similar to knPCA. Kernel PCA is computed by performing an eigenvalue decomposition of a kernel version of the covariance matrix. When using the linear kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}, \mathbf{y})$, kernel PCA reduces to PCA. Thus kernel PCA is a direct generalization of PCA and requires only linear optimization. In fact, the only difference between kernel PCA and knPCA is that knPCA uses nPCA to approximate PCA before using an SVM kernel. This gives knPCA two advantages over kernel PCA. First, the nonlinear principal components found by kernel PCA do not, in general, correspond to points in the original data set. In fact, a kernel PCA component may not correspond to *any* point in the original data *space*. In comparison, each nonlinear principal component found by knPCA corresponds directly to a point in the original data set. This is an advantage when interpreting the principal components that knPCA finds. Second, the computation of kernel PCA requires an eigenvalue decomposition of an $n \times n$ matrix, where n is the number of data points under consideration. Although linear, this is a difficult calculation for large n. In comparison, knPCA is matrix free. KnPCA uses the same matrix, but the explicit formulation of the matrix is not required. Even if the matrix is formed, an eigenvalue decomposition is not necessary.

7 Extensions

In this paper we presented knPCA as a combination of Gram-Schmidt and PCA. We modified Gram-Schmidt and used the PCA criterion to select the linearly independent subset required by Gram-Schmidt. Of course, there is no reason to constrain ourselves to the PCA criterion. Any criterion that can be written in terms of inner products will work. In [17] we used points with maximal norm to arrive at our original kernel version of Gram-Schmidt. This algorithm, kernel Gram-Schmidt, is faster than knPCA (since most of the time spent by knPCA is spent computing variances), but produces a more generic representation of the data. Other useful criterion might provide clustering representations or representations using independent components.

Another way to extend the usefulness of knPCA is by combining it with other algorithms. This might be more useful in practice with kernel Gram-Scmidt than with knPCA but the idea is the same: perform knPCA, apply any algorithm (probably linear), then translate back to the original space using the change of basis matrix T (see Section 3) to rewrite the results in terms of the original data points. We executed the first two steps of this sequence on the handwritten digit data in Section 5, where we used a linear SVM in the second step. It might be interesting to apply different classification algorithms here as well as algorithms for regresssion.

8 Conclusion

We have introduced both an interesting appoximation to PCA and a nonlinear extension of that approximation. NPCA performs PCA constrained to the original data set. This makes the singular vectors found by nPCA more interpretable than the analagous singular vectors found by standard PCA. Thus nPCA provides information that is difficult to obtain by PCA alone. This was illustrated using the Taylor-Couette simulation data in Section 5.

KnPCA provides a nonlinear generalization of nPCA and so can be considered a nonlinear generalization of PCA. KnPCA provides the interpretability of nPCA in a nonlinear setting. In Section 5 this was shown to be useful for interpreting the actions of nonlinear SVMs on the USPS database of handwritten digits.

The algorithm which performs knPCA is also capable of data analysis on very large data sets. It is parallelizable and can be implemented to exploit computers with large memory capacities. (It can also be used with minimal memory requirements.)

In addition, the knPCA algorithm is modular and can be easily changed to perform other tasks. It can be extended for use with other algorithms such as regression and classification. All told, knPCA appears to be a very useful and adaptable tool for data analysis.

References

[1] R. Adair. *Simulations of Taylor-Couette Flow.* PhD thesis, Colorado State University, 1997.

[2] AT&T database of faces. http://www.uk.research.att.com/facedatabase.html, 2001. AT&T Laboratories, Cambridge, UK.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In D. Haussler, editor, *5th Annual ACM Workshop on COLT*, pages 144–152, Pittsburgh, PA, 1992. ACM Press.

[4] C. J. C. Burges. A Tutorial on Support Vector Machines for Pattern Recognition. *Knowledge Discovery and Data Mining*, 2(2), 1998.

[5] C. J. C. Burges. Geometry and invariance in kernel based methods. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 89–116, Cambridge, MA, 1999. MIT Press.

[6] C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273 – 297, 1995.

[7] K.I. Diamantaras and S.Y. Kung. *Principal Component Neural Networks.* John Wiley & Sons, 1996.

[8] J. H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82:249–266, 1987.

[9] T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84:502–516, 1989.

[10] H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441 and 498–520, 1933.

[11] Aapo Hyvärinen. Survey on Independent Component Analysis. *Neural Computing Surveys*, 2:94–128, 1999.

[12] T. Joachims. Making large–scale SVM learning practical. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods — Support Vector Learning*, pages 169–184, Cambridge, MA, 1999. MIT Press.

[13] I.T. Jolliffe. *Principal Component Analysis.* Springer-Verlag, 1986.

[14] C. Jutten and J. Herault. Blind separation of sources, part I: An adaptive algorithm based on neuromimetic architecture, 1991.

[15] Michael Kirby. *Geometric Data Analysis.* John Wiley & Sons, 2001.

[16] M. A. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37:233–243, 1991.

[17] S. Martin, M. Kirby, and R. Miranda. Kernel/feature selection for support vector machines applied to materials design. In *IFAC Symposium on Artificial Intelligence in Real Time Control AIRTC-2000*, pages 29–34. Elsevier Science, Ltd., 2000.

[18] E. Oja. A simplified neuron model as a principal component analyzer. *Journal of Mathematical Biology*, 15(3):267–273, 1982.

[19] K. Pearson. On lines and planes of closest fit to systems of points in space. *Phil. Mag.*, 2:559–572, 1901.

[20] F. Samaria and A. Harter. Parameterisation of a stochastic model for human face identification. In *2nd IEEE Workshop on Applications of Computer Vision*, 1994.

[21] B. Schölkopf. *Support Vector Learning.* Oldenbourg Verlag, Munich, 1997. Doktorarbeit, TU Berlin.

[22] B. Schölkopf. kpca_toy.m `MATLAB` software. http://www.kernel-machines.org, 1998.

[23] B. Schölkopf, C. Burges, and V. Vapnik. Extracting support data for a given task. In U. M. Fayyad and R. Uthurusamy, editors, *Proceedings, First International Conference on Knowledge Discovery & Data Mining*, Menlo Park, 1995. AAAI Press.

[24] B. Schölkopf, A. Smola, and K.-R. Müller. Kernel principal component analysis. In B. Schölkopf, C. J. C. Burges, and A. J. Smola, editors, *Advances in Kernel Methods - SV Learning*, pages 327–352. MIT Press, Cambridge, MA, 1999.

[25] J. Thomas. Taylor-Couette simulation data. Colorado State University, 1998. Private Communication.

[26] L. Trefethen and D. Bau. *Numerical Linear Algebra.* SIAM, Philadelphia, 1997.

[27] United States Postal Service database of handwritten digits. http://www.kernel-machines.org, 1991.

[28] V. Vapnik. *Statistical Learning Theory.* Wiley Interscience, New York, 1998.

Distribution

1	MS 1110	D. Womble, 9214
1	MS 0310	M. Rintoul, 9212
1	MS 0318	G. S. Davidson, 9212
1	MS 1110	W. E. Hart, 9211
5	MS 0318	S. Martin, 9212
1	MS 9018	Central Technical Files, 8945-1
2	MS 0899	Technical Library, 9616
1	MS 0612	Review and Approval Desk for DOE/OSTI, 9612